

Machine Learning

Supervised Learning

1. target / outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables).
2. Using these set of variables, we generate a function that map inputs to desired outputs.
3. The training process continues until the model achieves a desired level of accuracy on the training data.

Examples of Supervised Learning:

Regression, [Decision Tree](#), [Random Forest](#), KNN, Logistic Regression etc.

Unsupervised Learning

1. not have any target or outcome variable to predict / estimate.
2. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention.

Examples of Unsupervised Learning:

K-means clustering, Dimensionality Reduction.

List of Common Machine Learning Algorithms

1. Linear Regression
2. Logistic Regression
3. Decision Tree
4. Random Forest
5. Gradient Boosting
6. SVM
7. Naive Bayes
8. kNN
9. K-Means
10. Dimensionality Reduction Algorithms

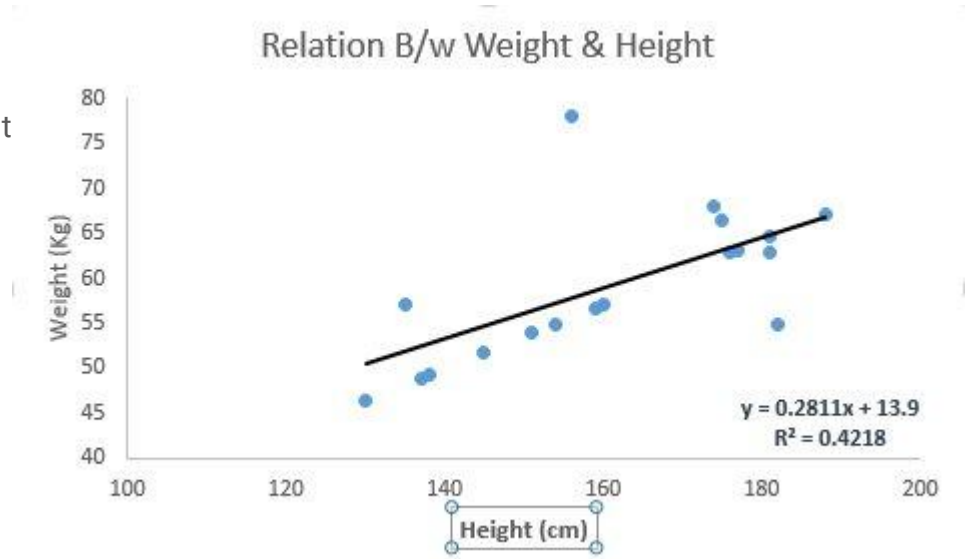
1. Linear Regression

1. It is used to estimate real values based on continuous variable(s).
2. Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation $Y = a * X + b$.

In this equation:

- Y – Dependent Variable
- a – Slope
- X – Independent variable
- b – Intercept

These coefficients a and b are derived based on minimizing the sum of squared difference of distance between data points and regression line.



Basic concepts

Step 1: Compute the Loss

Step 2: Compute the Gradients

Step 3: Update the Parameters

Step 1: Compute the Loss

For a regression problem, the **loss** is given by the **Mean Square Error (MSE)**, that is, the average of all squared differences between **labels** (y) and **predictions** ($a + bx$)

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - a - bx_i)^2$$

Step 2: Compute the Gradients

Gradient descent is an optimization algorithm used to minimize loss function by iteratively moving in the direction of **steepest descent** as defined by the negative of the **gradient**.

In machine learning, **we** use **gradient descent** to update the parameters of our model.

$$\frac{\partial MSE}{\partial a} = \frac{\partial MSE}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial a} = \frac{1}{N} \sum_{i=1}^N 2(y_i - a - bx_i) \cdot (-1) = -2 \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)$$

$$\frac{\partial MSE}{\partial b} = \frac{\partial MSE}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial b} = \frac{1}{N} \sum_{i=1}^N 2(y_i - a - bx_i) \cdot (-x_i) = -2 \frac{1}{N} \sum_{i=1}^N x_i (y_i - \hat{y}_i)$$

Step 3: Update the Parameters

In the final step, we **use the gradients to update** the parameters. Since we are trying to **minimize** our **losses**, we **reverse the sign** of the gradient for the update.

learning rate, denoted by the *Greek letter **eta*** (that looks like the letter **n**), which is the **multiplicative factor** that we need to apply to the gradient for the parameter update.

$$a = a - \eta \frac{\partial MSE}{\partial a}$$

$$b = b - \eta \frac{\partial MSE}{\partial b}$$

Step 4: Rinse and Repeat!

Now we use the **updated parameters** to go back to **Step 1** and restart the process. Repeating this process over and over, for **many epochs**.

Small Hint:

***batch** gradient descent Vs **mini-batch** gradient descent*

- *An **epoch** is complete whenever every point has been already used for computing the loss.*
- *For **batch** gradient descent, this is trivial, as it uses all points for computing the loss — **one epoch** is the same as **one update**.*
- *For **stochastic** gradient descent, **one epoch** means **N updates**, while for **mini-batch** (of size n), **one epoch** has **N/n updates**.*

scikit-learn

```
from sklearn.linear_model import LinearRegression
```

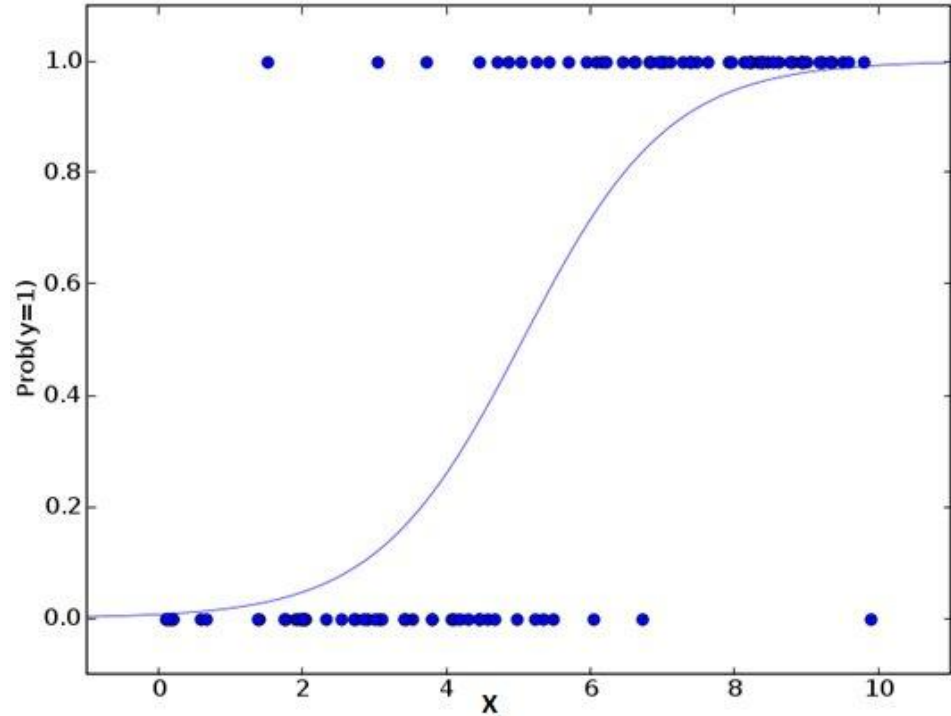
```
LinearReg = LinearRegression()
```

```
LinearReg.fit(X1, y1)
```

```
LinearReg.predict(X1)
```

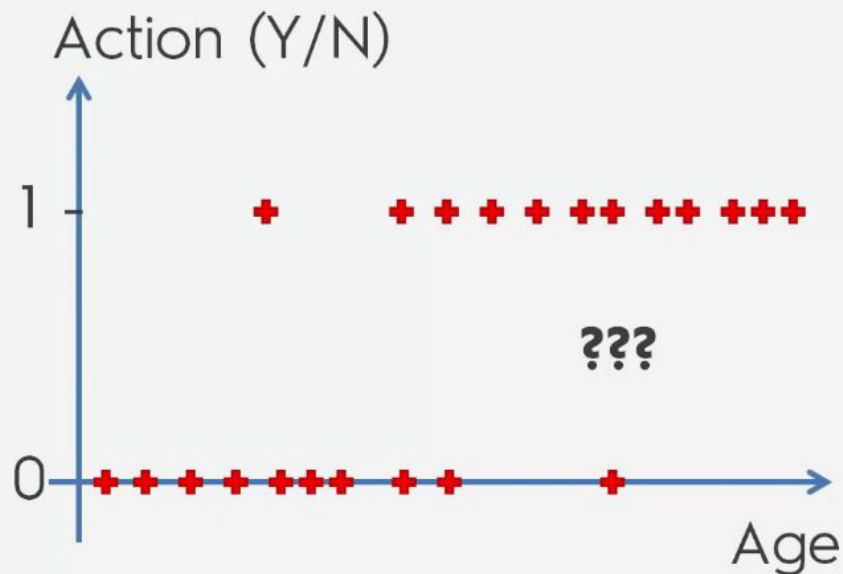
2. Logistic Regression

1. It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false) based on given set of independent variable(s).
2. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Hence, it is also known as **logit regression**.
3. Since, it predicts the probability, its output values lies between 0 and 1 (as expected)..

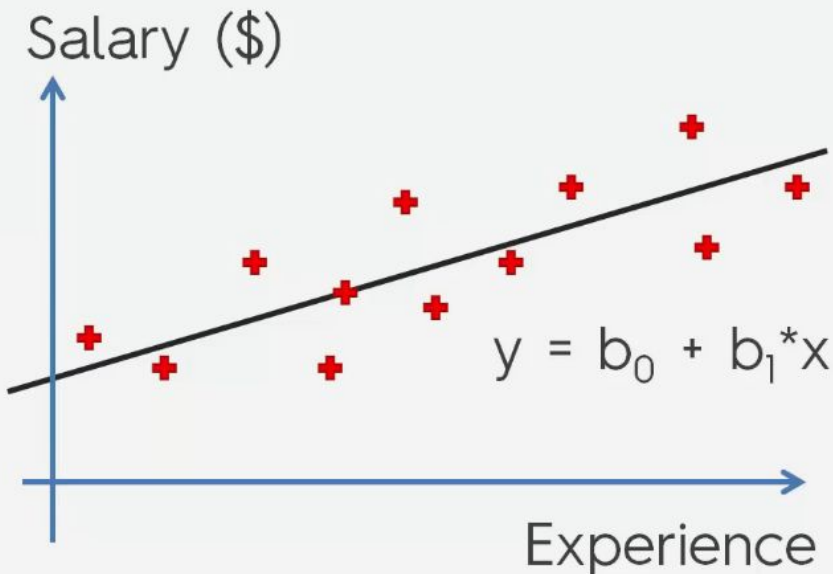


Logistic Regression

This is new:



We know this:



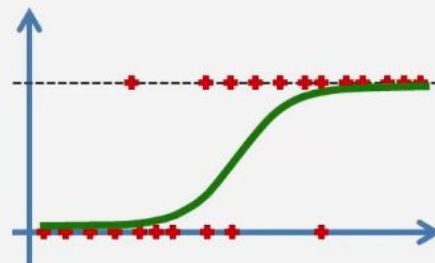
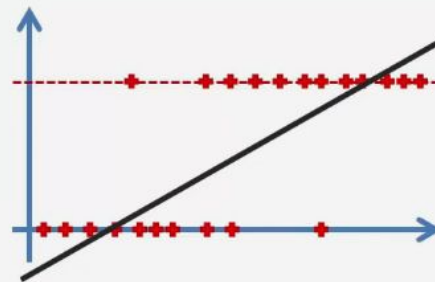
Logistic Regression

$$y = b_0 + b_1 * x$$

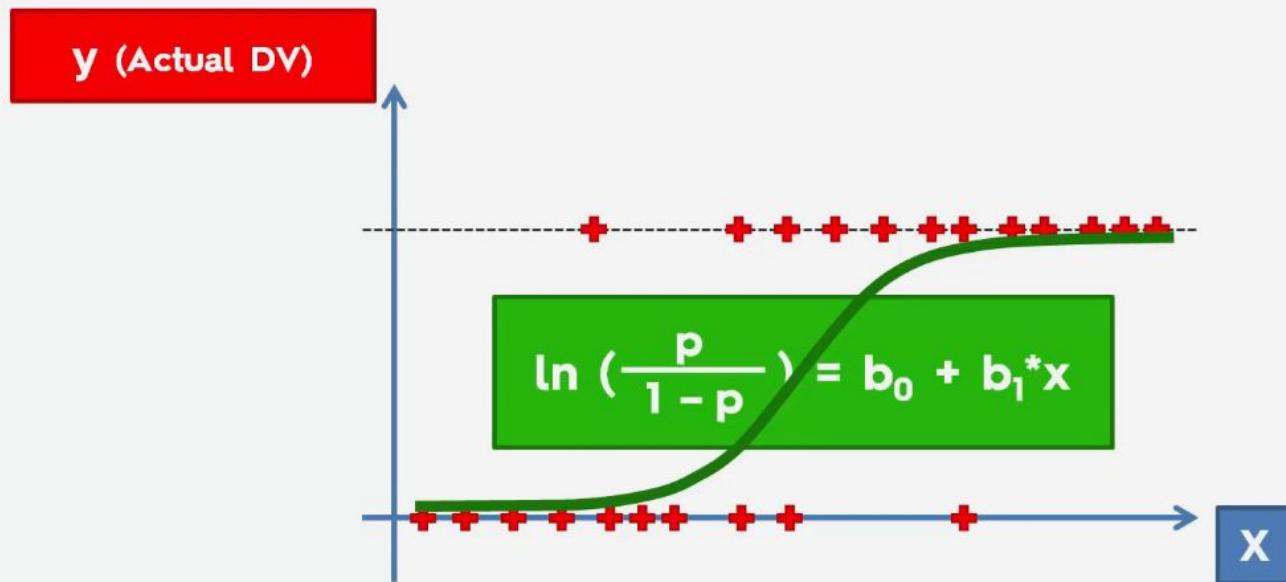
Sigmoid Function

$$p = \frac{1}{1 + e^{-y}}$$

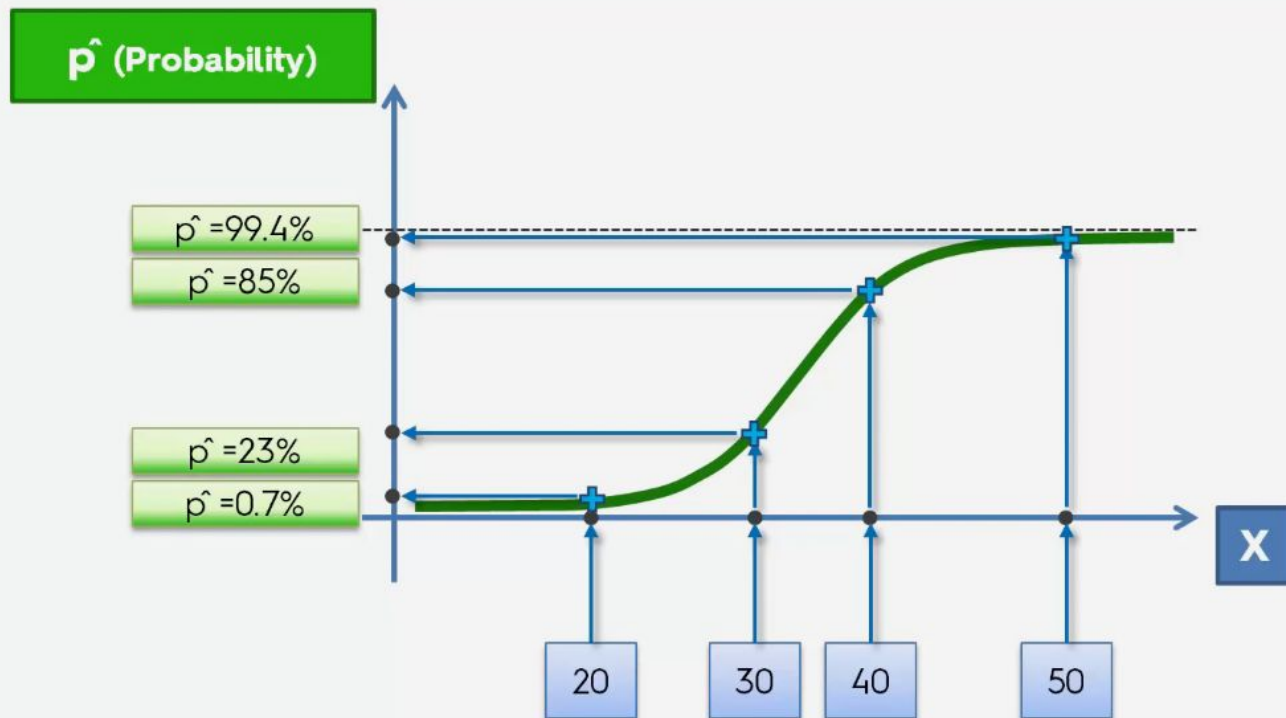
$$\ln \left(\frac{p}{1 - p} \right) = b_0 + b_1 * x$$



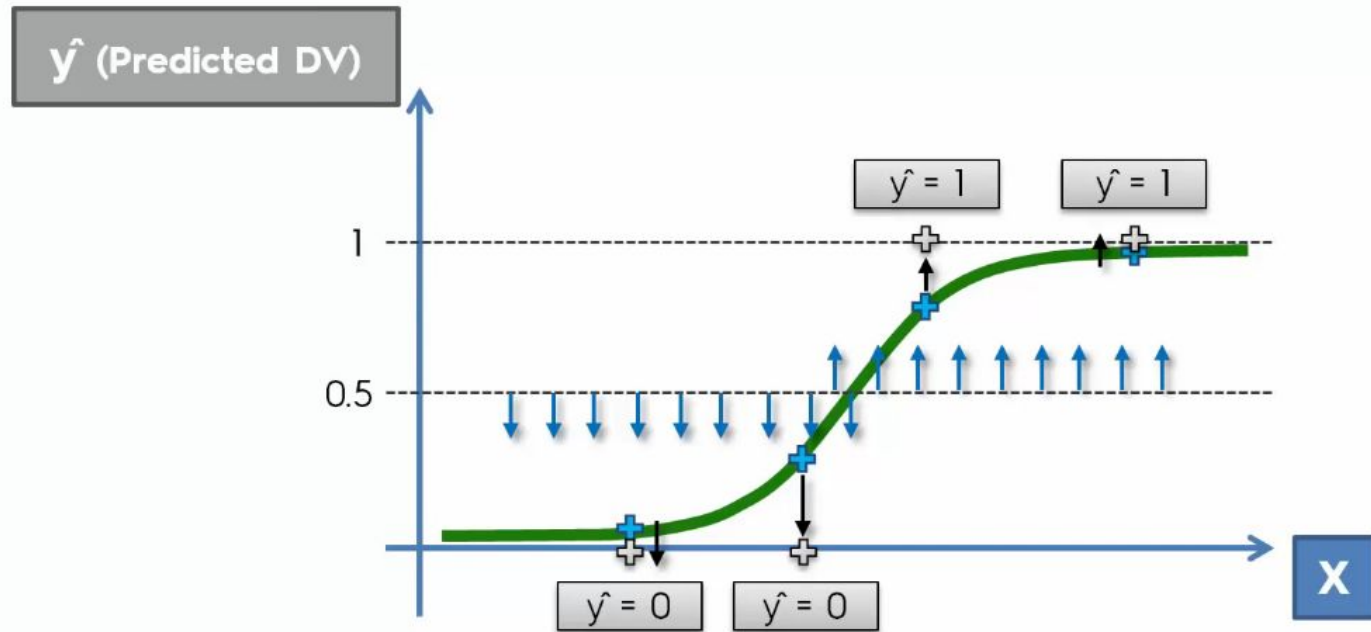
Logistic Regression



Logistic Regression



Logistic Regression



scikit-learn

```
from sklearn.linear_model import LogisticRegression
```

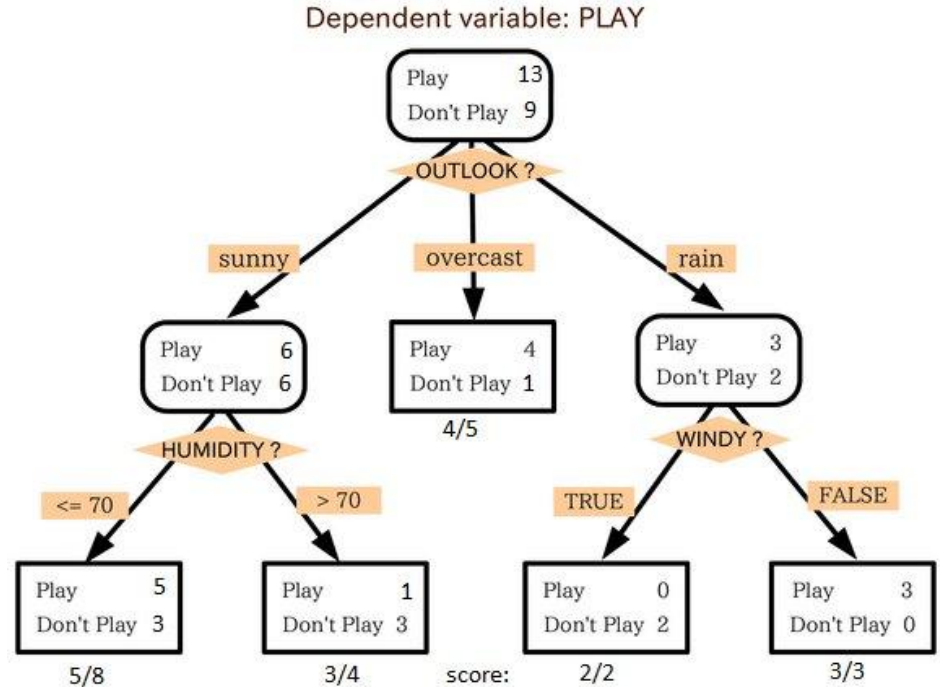
```
LogisticReg = LogisticRegression(random_state=0)
```

```
LogisticReg.fit(X_train, y_train)
```

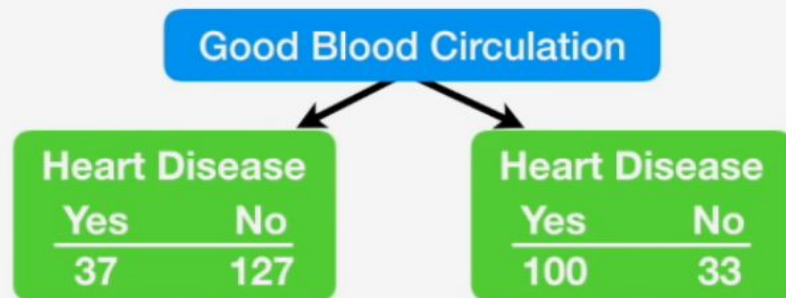
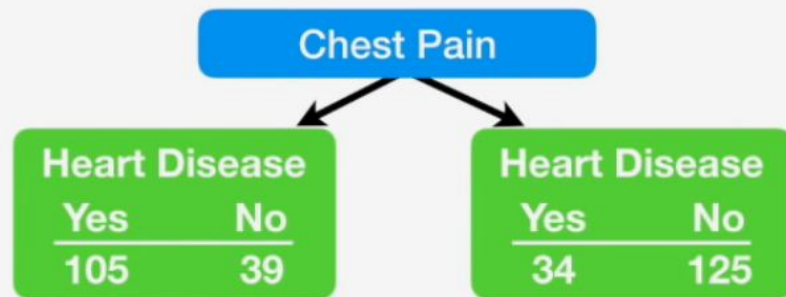
```
LogisticReg.predict(X_test)
```

3. Decision Tree

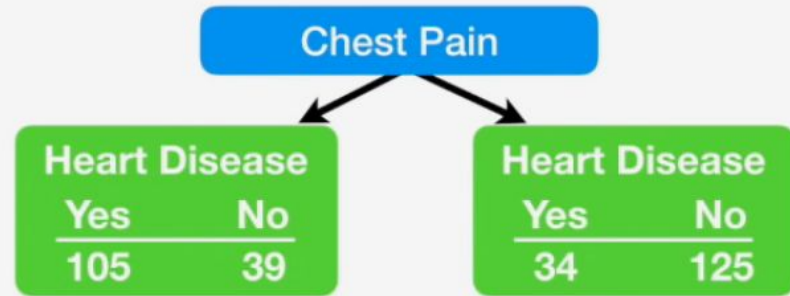
1. It is a type of supervised learning algorithm that is mostly used for classification problems.
2. Surprisingly, it works for both categorical and continuous dependent variables.
3. In this algorithm, we split the population into two or more homogeneous sets.
4. This is done based on most significant attributes/independent variables to make as distinct groups as possible..



Chest Pain	Good Blood Circulation	Blocked Arteries	Heart Disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
etc...	etc...	etc...	etc...



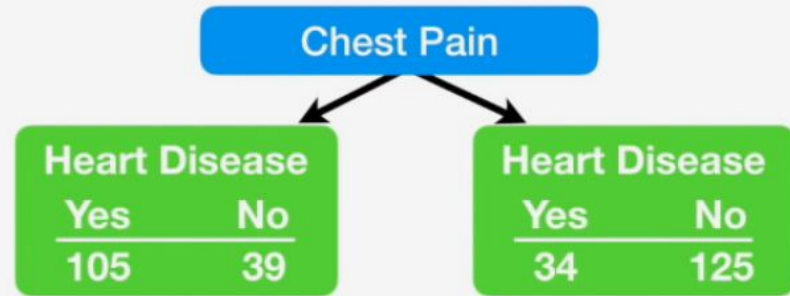
Then we looked at how well
Good Blood Circulation
 separated patients with and
 without heart disease.



For this leaf, the Gini impurity = $1 - (\text{the probability of "yes"})^2 - (\text{the probability of "no"})^2$

$$= 1 - \left(\frac{105}{105 + 39} \right)^2 - \left(\frac{39}{105 + 39} \right)^2$$

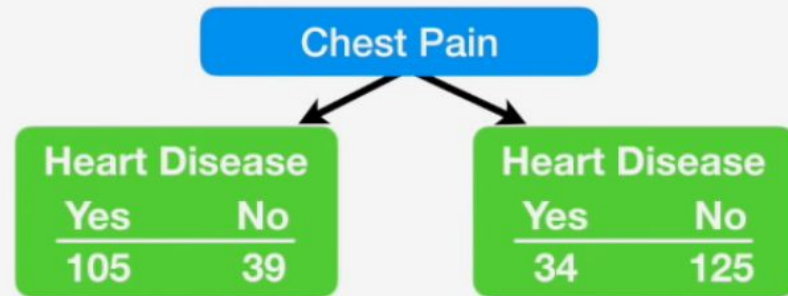
$$= 0.395$$



$$= 1 - (\text{the probability of "yes"})^2 - (\text{the probability of "no"})^2$$

$$= 1 - \left(\frac{34}{34 + 125} \right)^2 - \left(\frac{125}{34 + 125} \right)^2$$

$$= 0.336$$



Gini impurity = 0.395

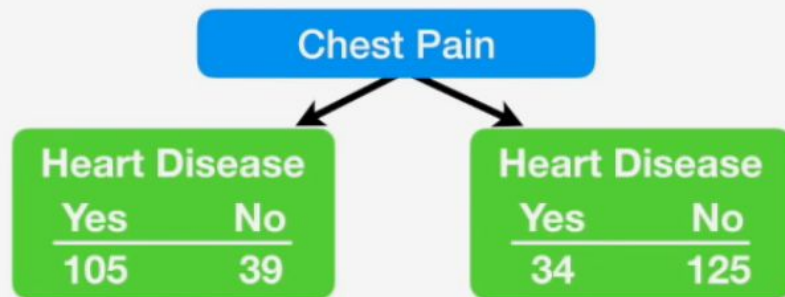
0.336

Gini impurity for Chest Pain = weighted average of Gini impurities for the leaf nodes

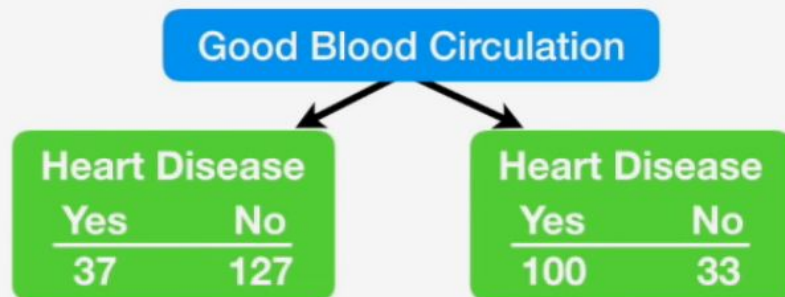
$$= \left(\frac{144}{144 + 159} \right) 0.395 + \left(\frac{159}{144 + 159} \right) 0.336$$

$$= 0.364$$

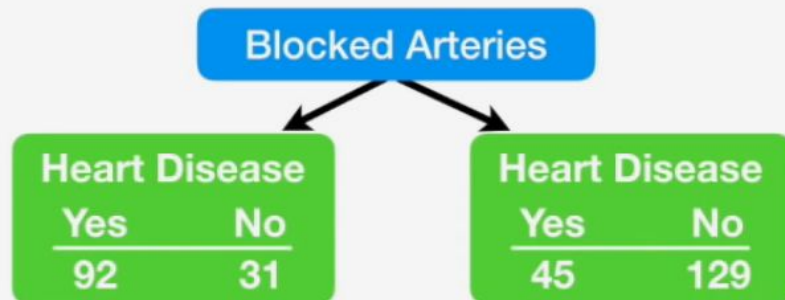
Gini impurity for Chest Pain = 0.364

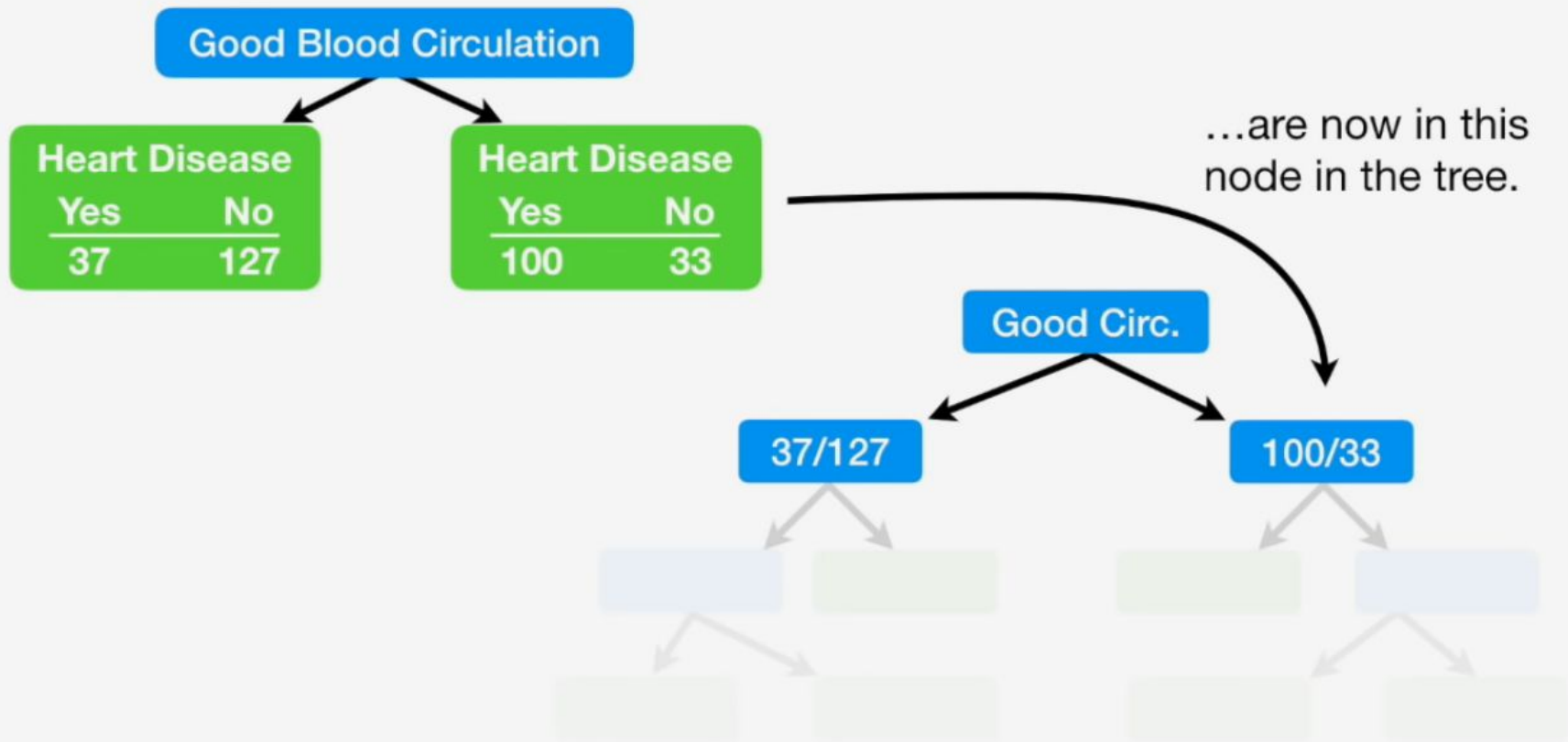


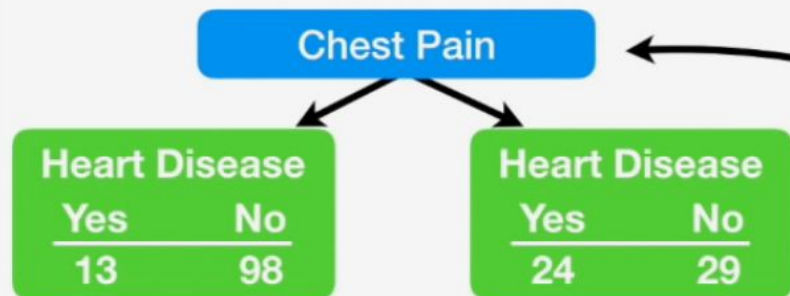
Gini impurity for Good Blood Circulation = 0.360



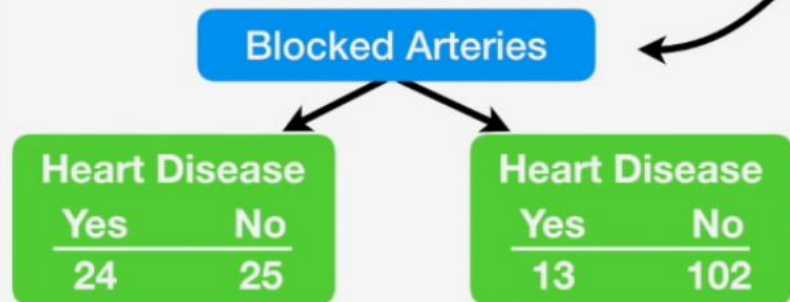
Gini impurity for Blocked Arteries = 0.381



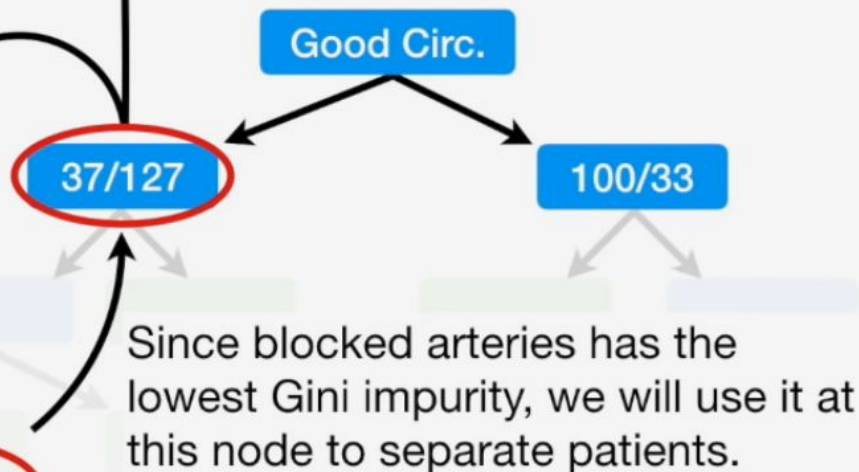


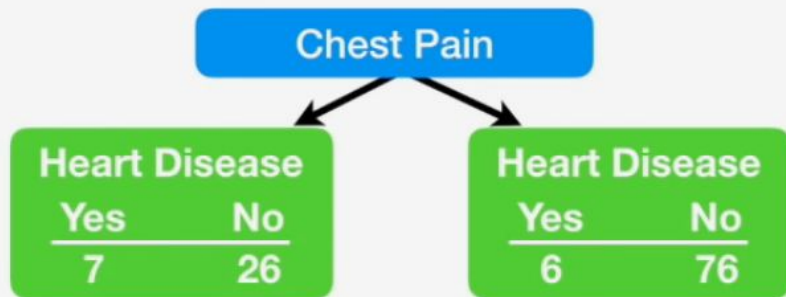


Gini impurity for Chest Pain = 0.3



Gini impurity for Blocked Arteries = 0.290





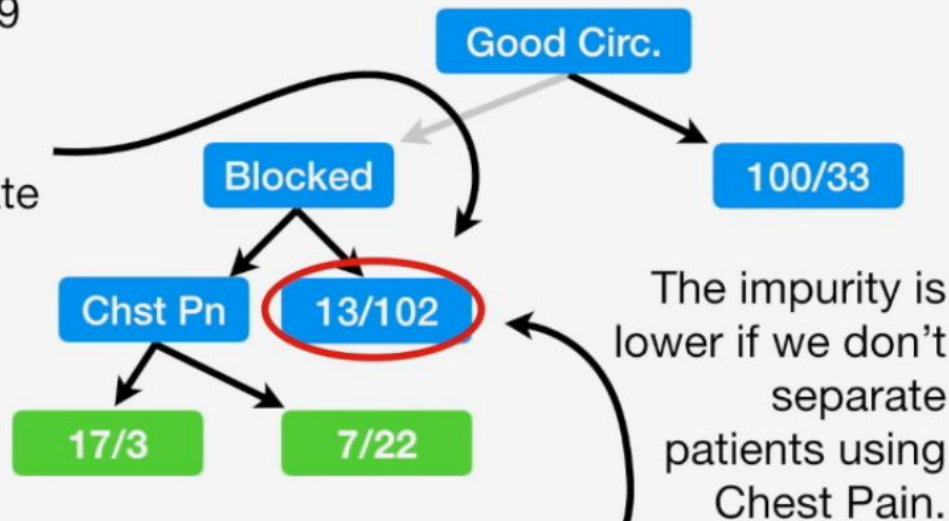
Gini impurity for Chest Pain = 0.29

The Gini impurity for this node, before using chest pain to separate patients is...

$$= 1 - (\text{the probability of "yes"})^2 - (\text{the probability of "no"})^2$$

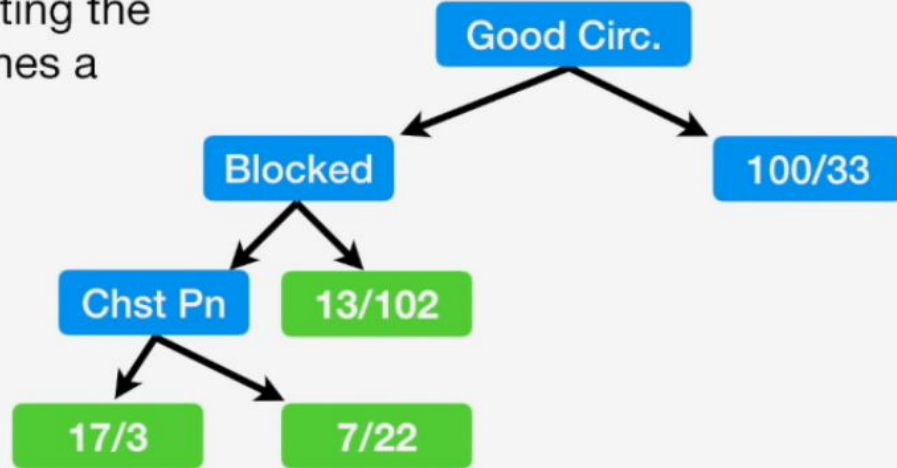
$$= 1 - \left(\frac{13}{13 + 102}\right)^2 - \left(\frac{102}{13 + 102}\right)^2$$

$$= 0.2$$

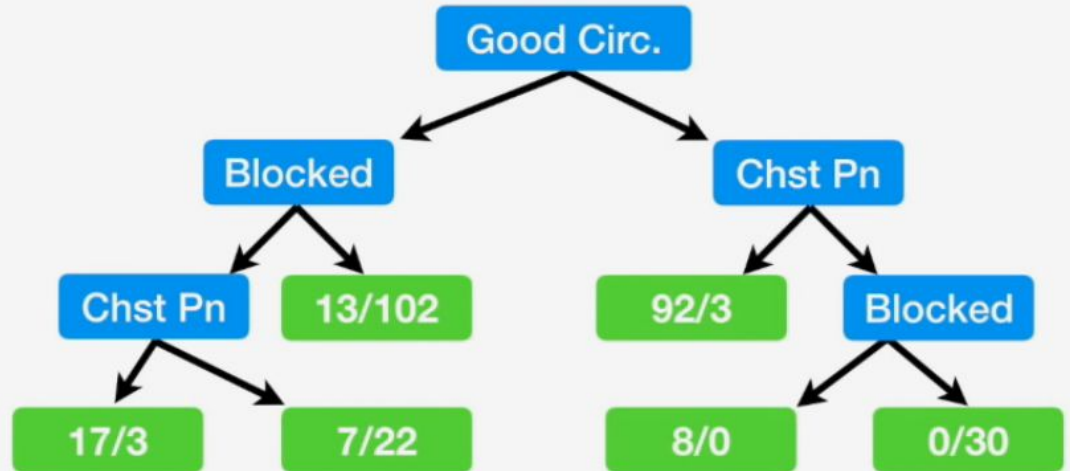


The good news is that we follow the exact same steps as we did on the left side:

- 1) Calculate all of the Gini impurity scores.
- 2) If the node itself has the lowest score, than there is no point in separating the patients any more and it becomes a leaf node.
- 3) If separating the data results in an improvement, than pick the separation with the lowest impurity value.



Hooray!!! We made a decision tree!!!



	toothed	breathes	legs	species
0	True	True	True	Mammal
1	True	True	True	Mammal
2	True	True	False	Reptile
3	False	True	True	Mammal
4	True	True	True	Mammal
5	True	True	True	Mammal
6	True	False	False	Reptile
7	True	True	False	Reptile
8	True	True	True	Mammal
9	False	True	True	Reptile

toothed == True

toothed == False

	toothed	breathes	legs	species
0	True	True	True	Mammal
1	True	True	True	Mammal
2	True	True	False	Reptile
4	True	True	True	Mammal
5	True	True	True	Mammal
6	True	False	False	Reptile
7	True	True	False	Reptile
8	True	True	True	Mammal

	toothed	breathes	legs	species
3	False	True	True	Mammal
9	False	True	True	Reptile

After computing the IG of feature *toothed*
do this for features *breathes* and *legs*

1. Calculate the entropy
for *toothed* == True



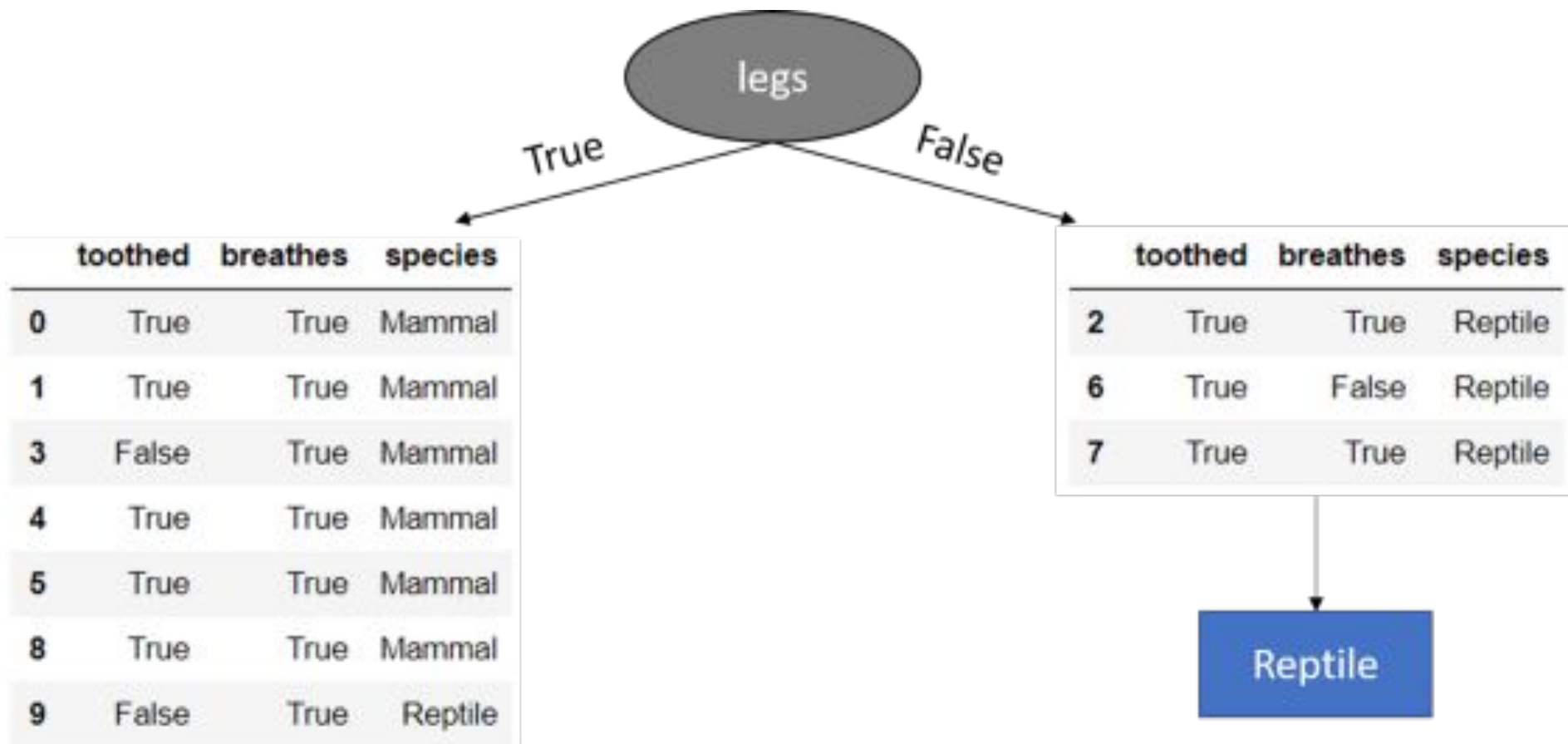
2. Calculate the entropy
for *toothed* == False

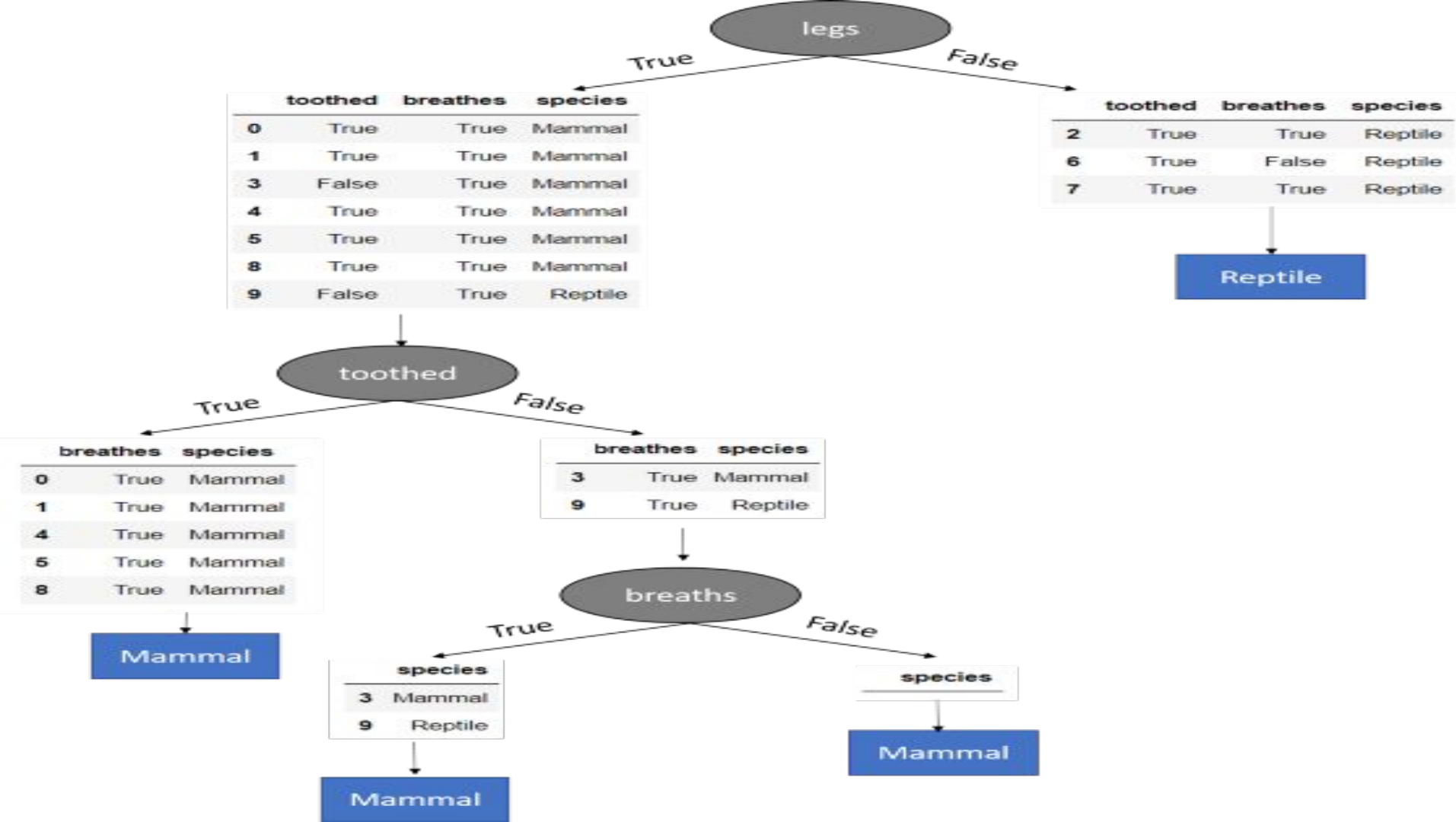


3. Sum up the entropies
of 1. and 2.



4. Subtract this sum
from the whole datasets
entropy → InfoGain





scikit-learn

```
from sklearn.tree import DecisionTreeClassifier
```

```
DT_Model = DecisionTreeClassifier(random_state=0)
```

```
DT_Model.fit(X_train, y_train)
```

```
DT_Model.predict(X_test)
```

Random forest

1. **Random forest** algorithm can be used for both classifications and regression task.
2. It provides higher accuracy. **Random forest** classifier will handle the missing values and maintain the accuracy of a large proportion of data.
3. If there are more trees, it won't allow overfitting trees in the model.

Ada Boost

When nothing works, Boosting does. Nowadays many people use either XGBoost or LightGBM or CatBoost to win competitions at Kaggle or Hackathons. AdaBoost is the first stepping stone in the world of Boosting.

AdaBoost is one of the first boosting algorithms to be adapted in solving practices.

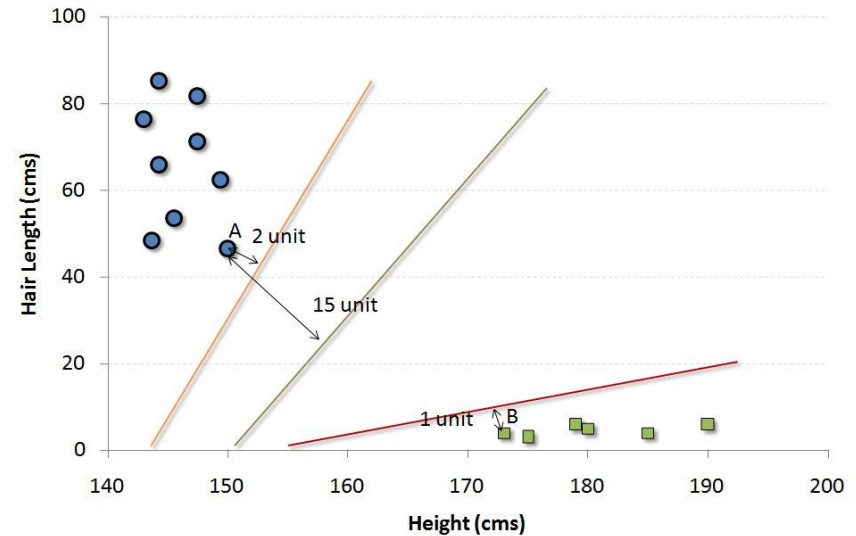
1. Adaboost helps you **combine multiple “weak classifiers” into a single “strong classifier”**.
2. The weak learners in AdaBoost are decision trees with a single split, called decision stumps.
3. AdaBoost works by putting more weight on difficult to classify instances and less on those already handled well.
4. AdaBoost algorithms can be used for both classification and regression problem.

4. SVM (Support Vector Machine)

It is a classification method. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.

For example, if we only had two features like Height and Hair length of an individual, we'd first plot these two variables in two dimensional space where each point has two co-ordinates (these co-ordinates are known as **Support Vectors**)

1. Hyperplane
2. Marginal value
3. Support vector



scikit-learn

```
from sklearn.svm import SVC
```

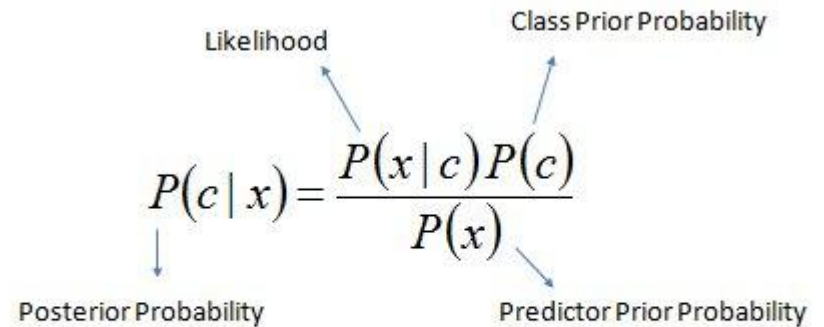
```
SVM_Model = SVC()
```

```
SVM_Model.fit(X_train, y_train)
```

```
SVM_Model.predict(X_test)
```

5. Naive Bayes

1. It is a classification technique based on Bayes' theorem with an assumption of independence between predictors.
2. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.



The diagram shows the Naive Bayes formula with arrows pointing from labels to the corresponding parts of the equation:

$$P(c | x) = \frac{P(x | c)P(c)}{P(x)}$$

- Likelihood** points to $P(x | c)$
- Class Prior Probability** points to $P(c)$
- Posterior Probability** points to $P(c | x)$
- Predictor Prior Probability** points to $P(x)$

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$$

scikit-learn

```
from sklearn.naive_bayes import GaussianNB
```

```
NB_Model = GaussianNB()
```

```
NB_Model.fit(X_train, y_train)
```

```
NB_Model.predict(X_test)
```


6. kNN (k- Nearest Neighbors)

1. The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.
2. These distance functions can be Euclidean, Manhattan, are used for continuous function
3. If $K = 1$, then the case is simply assigned to the class of its nearest neighbor. At times, choosing K turns out to be a challenge while performing kNN modeling.

scikit-learn

```
from sklearn.neighbors import KNeighborsClassifier
```

```
KNN_Model = KNeighborsClassifier(n_neighbors=3)
```

```
KNN_Model.fit(X_train, y_train)
```

```
KNN_Model.predict(X_test)
```

7. K-Means

1. It is a type of unsupervised algorithm which solves the clustering problem.
2. Its procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters).
3. Data points inside a cluster are homogeneous and heterogeneous to peer groups.

scikit-learn

```
from sklearn.cluster import KMeans
```

```
KmeansModel = KMeans(n_clusters=3, random_state=0)
```

```
KmeansModel.fit(X_train)
```

```
KmeansModel.predict(X_test)
```

9. Dimensionality Reduction Algorithms

dimensionality reduction algorithm helps us to reduce the feature dimensions

Principal Component Analysis

scikit-learn

```
from sklearn.decomposition import PCA
```

```
PCA_Model = PCA(n_components=3)
```

```
PCA_Model.fit(X_train)
```

```
train_set = PCA_Model.transform(X_train)
```

reference

<https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/>